1.Put VS Patch?

Video Link : https://www.youtube.com/watch?v=LJajkjI5RHE&t=499s

When a client needs to replace an existing Resource entirely, they can use PUT. When they're doing a partial update, they can use HTTP PATCH.

For instance, when updating a single field of the Resource, sending the complete Resource representation can be cumbersome and uses a lot of unnecessary bandwidth. In such cases, the semantics of PATCH make a lot more sense.

What is PUT

PUT is another HTTP method used to create a new resource at a specified URI or to update an existing resource. Although PUT can be used to create a resource, it is most often used to update resource. To create a new resource, using PUT we need to know the exact URI where the data needs to be put. If incase there is data in the specified URI, the entire data is overwritten which is update.

```
{
"id":5
    "blog_title": "What is PUT",
    "blog_post": "Confused whether to use PUT or PATCH or POST...",
    "blog_post_author": "programmerspub"
}
```

To create a resource at specified URI /5:

```
HTTP PUT  https://programmerspub.com/blog/5
```

```
Payload:
```

```
{
    "blog_title": "What is PUT",
    "blog_post": "Confused whether to use PUT or PATCH or POST...",
    "blog_post_author": "programmerspub"
}
```

To update a resource:

```
HTTP PUT  https://programmerspub.com/blog/5
```

```
Payload:
```

```
{
    "blog_title": "What is PUT",
    "blog_post": "Confused whether to use PUT or PATCH or
POST...updated post...",
    "blog_post_author": "programmerspub"
}
```

Most important thing to remember is the PUT payload should have all the fields of the resource when updating or else the resource is overwritten with the data we send.

For example consider this PUT request where we send only the updated blog title.

```
HTTP PUT  https://programmerspub.com/blog/general/5
```

```
Payload:
```

```
{
    "blog_title": "What is PUT - updated"
}
```

After the PUT request only the blog title has the updated value while blog_post and blog_post_author fields are lost.

```
Response:
{
    "blog_title": "What is PUT - updated"
}
```

What is PATCH:

PATCH is another HTTP method which is used to update a resource with partial data. Unlike PUT, PATCH does not need the full payload to update a resource. For example if a resource has 100 fields, using PATCH would be a better option than PUT as PUT requires all 100 fields to be sent again to update a resource.

Existing Resource:

```
{
    "blog_title": "What is PATCH",
    "blog_post": "Confused whether to use PUT or PATCH or POST...",
    "blog_post_author": "programmerspub"
}
```

```
HTTP PATCH  https://programmerspub.com/blog/general/5
```

```
Payload:
```

```
{
    "blog_title": "What is PATCH - updated"
}
```

After the PATCH request, only the field blog_title is update while other fields remain intact.

```
{
    "blog_title": "What is PATCH - updated",
    "blog_post": "Confused whether to use PUT or PATCH or POST...",
    "blog_post_author": "programmerspub"
}
```

2.What are HTTP status codes?

Video Link : https://www.youtube.com/watch?v=Izh1yEhX60Y

HTTP status codes are three-digit responses from the server to the browser-side request.

These status codes (also called response status codes) serve as a means of communication between the server and the internet browser and there are multiple code classes based on the type of information they are communicating. The differences in classes are indicated through the first digit of the error code, for example: just like a 404, any other 4xx will mean that in some way the page or website could not be reached, while a 2xx means that your request was successfully completed.

HTTP status codes are divided into 5 “classes”. These are groupings of responses that have similar or related meanings. Knowing what they are can help you quickly determine the general substance of a status code before you go about looking up its specific meaning.

The five classes include:

- 100s: Informational codes indicating that the request initiated by the browser is continuing.
- 200s: Success codes returned when browser request was received, understood, and processed by the server.
- 300s: Redirection codes returned when a new resource has been substituted for the requested resource.
- 400s: Client error codes indicating that there was a problem with the request.
- 500s: Server error codes indicating that the request was accepted, but that an error on the server prevented the fulfillment of the request.
- Within each of these classes, a variety of server codes exist and may be returned by the server. Each individual code has a specific and unique meaning, which we'll cover in the more comprehensive list below.

200 Status Codes

This is the best kind of HTTP status code to receive. A 200-level response means that everything is working exactly as it should.

- 200: "Everything is OK." This is the code that is delivered when a web page or resource acts exactly the way it's expected to.
- 201: "Created." The server has fulfilled the browser's request, and as a result, has created a new resource.
- 202: "Accepted." The server has accepted your browser's request but is still processing it. The request ultimately may or may not result in a completed response.

400 Status Codes

At the 400 level, HTTP status codes start to become problematic. These are error codes specifying that there's a fault with your browser and/or request.

- 400: "Bad Request." The server can't return a response due to an error on the client's end. See our guide for resolving this error.
- 401: "Unauthorized" or "Authorization Required." This is returned by the server when the target resource lacks valid authentication credentials. You might see this if you've set up basic HTTP authentication using htpasswd.
- 403: "Access to that resource is forbidden." This code is returned when a user attempts to access something that they don't have permission to view. For example, trying to reach password-protected content without logging in might produce a 403 error.

- 404: “The requested resource was not found.” This is the most common error message of them all. This code means that the requested resource does not exist, and the server does not know if it ever existed.
- 405: “Method not allowed.” This is generated when the hosting server (origin server) supports the method received, but the target resource doesn’t.

500 Status Codes

500-level status codes are also considered errors. However, they denote that the problem is on the server’s end. This can make them more difficult to resolve.

- 500: “There was an error on the server and the request could not be completed.” This is generic code that simply means “internal server error”. Something went wrong on the server and the requested resource was not delivered. This code is typically generated by third-party plugins, faulty PHP, or even the connection to the database breaking. Check out our tutorials on how to fix the error establishing a database connection and other ways to resolve a 500 internal server error.

- 501: “Not Implemented.” This error indicates that the server does not support the functionality required to fulfill the request. This is almost always a problem on the web server itself, and usually must be resolved by the host. Check out our recommendations on how to resolve a 501 not implemented error.
- 502: “Bad Gateway.” This error code typically means that one server has received an invalid response from another, such as when a proxy server is in use. Other times a query or request will take too long, and so it is canceled or killed by the server and the connection to the database breaks.

3.What is Node Js? Difference b/w Javascript and Node Js?

Video Links:

- https://www.youtube.com/watch?v=_RSL3S3Anxg
- https://www.youtube.com/watch?v=B8KdllPwxBw

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

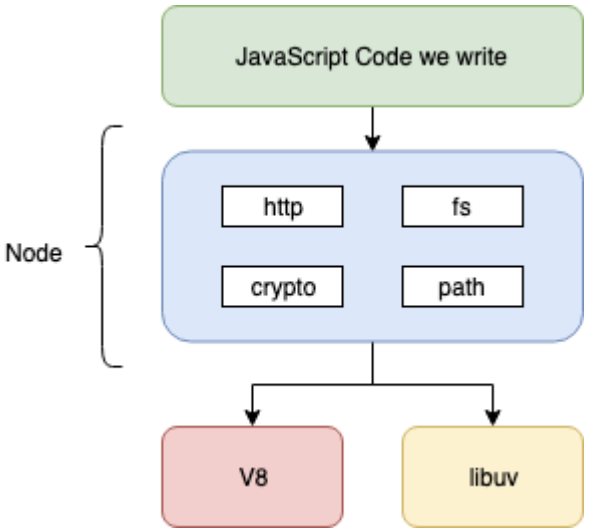
Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

What Is Node.js Written In?

Node.js is written in C, C++, and JavaScript.

Wikipedia defines Node.js as “a packaged compilation of Google’s V8 JavaScript engine, the libuv platform abstraction layer, and a core library, which is itself primarily written in JavaScript.”

The runtime uses Chrome V8 internally, which is the JavaScript execution engine, and it’s also written in C++. This adds additional use cases to Node.js’s repertoire, such as accessing internal system functionality (like networking).



Why Node.js?

Node.js has become the de facto tool for developing server-side and network applications. Here is why:

1. Node.js is really fast: Having been built on chrome V8 Javascript engine, its library is extremely fast for code execution.
2. Node Package Manager (NPM): Node Package Manager has more than 50,000 bundles, so whatever functionality is required for an application can be easily imported from NPM.
3. Node.js uses asynchronous programming: All APIs of Node.js library are asynchronous (i.e., non-blocking), so a Node.js-based server does not wait for the API to return data. The server calls the API, and in the event that no data is returned, the server moves to the next API the Events module of Node.js helps the server get a response from the previous API call. This also helps with the speed of Node.js.
4. No buffering: Node.js dramatically reduces the processing time while uploading audio and video files. Node.js applications never buffer data and simply output the data in chunks.

5. Single-threaded: Node.js makes use of a single-threaded model with event looping. As a result, it can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
6. Highly scalable: Node.js server responds in a non-blocking way, making it highly scalable in contrast with traditional servers, which create limited threads to handle requests.

These reasons more than justify the popularity of the Node.js platform and why it is being adopted by a large number of organizations and businesses. Now, let's familiarize ourselves with the different parts of Node.js.

Difference between Node.JS and Javascript

Let us talk about the differences between Node.JS and Javascript.

Parameters	Javascript	Node js
Basics	It is a programming language. We use JS mainly to write scripts on a website that makes web pages more dynamic in nature.	It is a runtime environment for Javascript that lets a user run this programming language on the server-side as well.
Running on Browsers	We can only run JS on browsers.	NodeJS helps us run JS outside the browser as well.
Client-side and Server-side	It is utilised on the web page’s client-side.	It lets us use JS on the server-side as well since it works on the server-side.
HTML Tags	The JS can easily add HTML and even play with the DOM.	The Node.JS, on the other hand, isn’t capable enough to add various HTML tags.

Mode of Running	We can run JS in any browser engine, such as the Spidermonkey in the Firefox browser and the JS core in the Safari browser.	Inside Node.JS, we have the JS engine known as V8. It helps in parsing and running the JS code.
Type of Development	It runs mainly on the client-side. Thus, it is used in the development of the front end.	It runs on the server-side. Thus, it helps in the server-side development via the JS.
Frameworks	Some very popular JS frameworks are TypedJS, RamdaJS, etc.	Some very commonly used Node.JS modules are Express, Lodash, etc. All of these modules need to be imported from the npm.
Writing of Script	The Javascript is nothing but the ECMA script's updated version that makes use of the Chrome V8 engine that is written in the C++ language.	C, C++, and also Javascript are used for writing Node.JS.

4.Node js is single threaded or multi threaded? Explain in detail what that means.

Video Link:

- https://www.youtube.com/watch?v=YSyFSnisip0
- https://www.youtube.com/watch?v=6YgsqXlUoTM

Event Loop Playground:

http://latentflip.com/loupe/?code=CmNvbnNvbGUubG9nKCJIaSEiKTsKCnNldFRpbWVvdXQoZnVuY3Rpb24gdGltZW91dCgpIHsKICAgIGNvbnNvbGUubG9nKCJDbGljayB0aGUgYnV0dG9uISIpOwp9LCA1MDAwKTsKCmNvbnNvbGUubG9nKCJXZWxjb21lIHRvIGxvdXBlLiIpOw%3D%3D!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D

What a Single-Threaded Process is

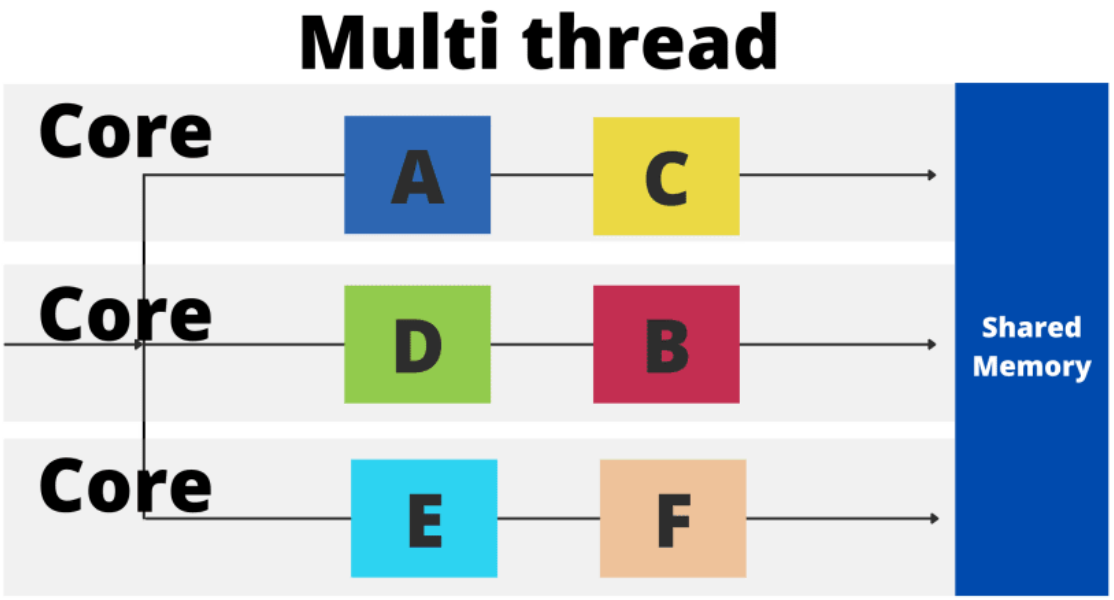
A single-threaded process is the execution of programmed instructions in a single sequence. Having said that, if an application has the following set of instructions:

- Instruction A
- Instruction B
- Instruction C

If these set of instructions are executed in a single-threaded process, the execution would look like the following:

What a Multi-Threaded Process is

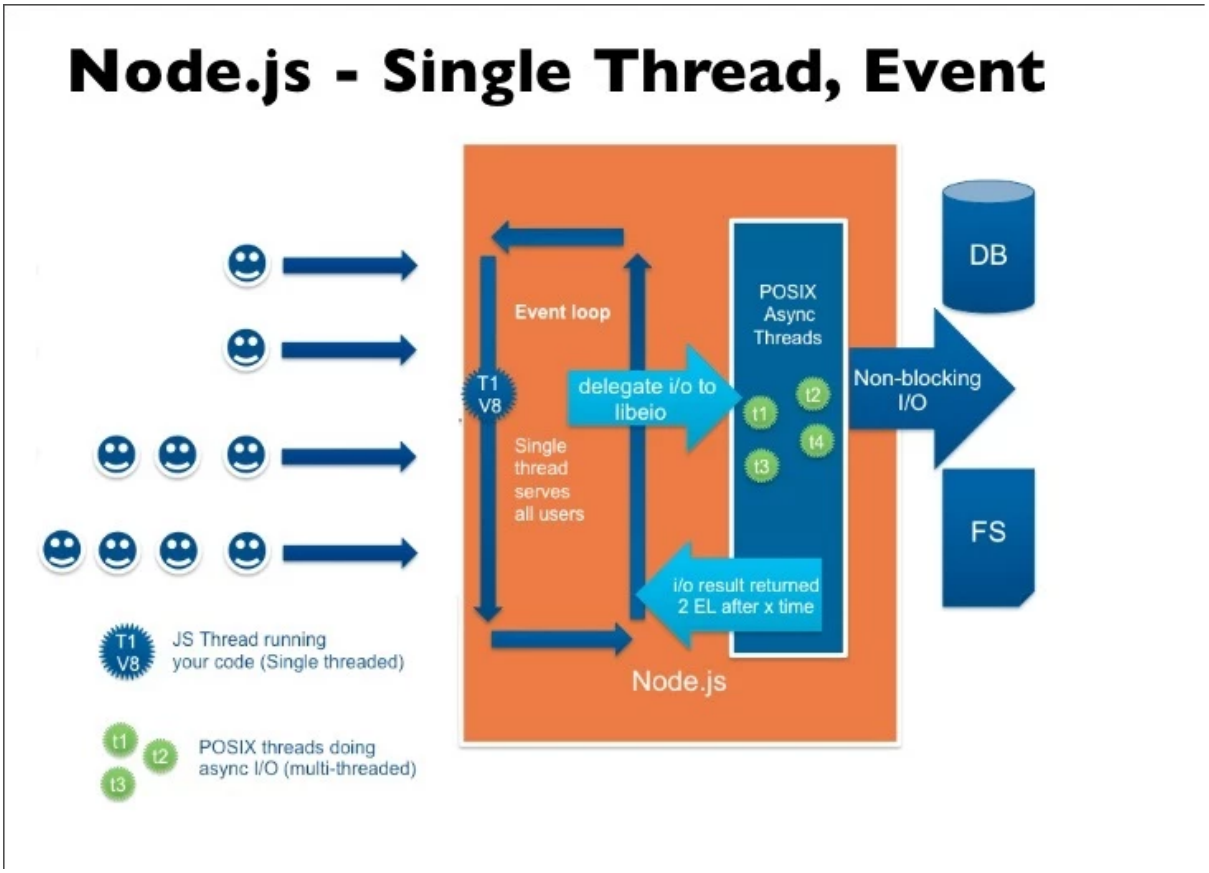
A multi-threaded process is the execution of programmed instructions in multiple sequences. Therefore, instructions won't have to wait to execute unless multiple instructions are grouped within different sequences.



Our Node.js applications are only sort of single-threaded, in reality. We can run things in parallel, but we don't create threads or sync them. The virtual machine and the operating system run the I/O in parallel for us, and when it's time to send data back to our JavaScript code, it's the JavaScript that runs in a single thread.

Node JS applications uses "Single Threaded Event Loop Model" architecture to handle multiple concurrent clients.

There are many web application technologies like JSP, Spring MVC, ASP.NET, HTML, Ajax, jQuery etc. But all these technologies follow "Multi-Threaded Request-Response" architecture to handle multiple concurrent clients.



Node JS Platform does not follow Request/Response Multi-Threaded Stateless Model. It follows Single Threaded with Event Loop Model. Node JS Processing model mainly based on Javascript Event based model with Javascript callback mechanism.

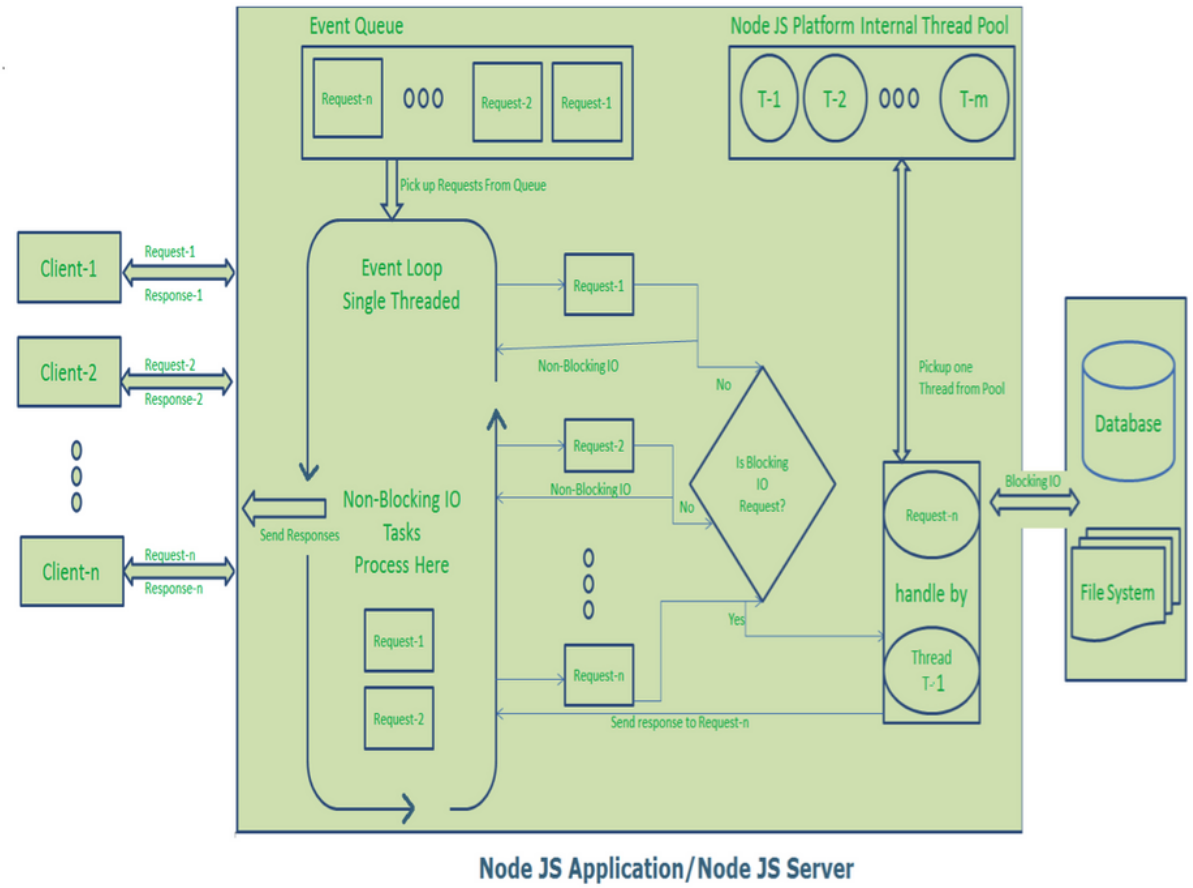
As Node JS follows this architecture, it can handle more and more concurrent client requests very easily.

The main heart of Node JS Processing model is “Event Loop”.

Here are Single Threaded Event Loop Model Processing Steps:

- Clients Send request to Web Server.
- Node JS Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.
- Node JS Web Server receives those requests and places them into a Queue. It is known as “Event Queue”.
- Event Loop uses Single Thread only. It is main heart of Node JS Platform Processing Model.

- Even Loop checks any Client Request is placed in Event Queue. If no, then wait for incoming requests for indefinitely.
- If yes, then pick up one Client Request from Event Queue
    1. Starts process that Client Request
    2. If that Client Request Does Not requires any Blocking IO Operations, then process everything, prepare response and send it back to client.
    3. If that Client Request requires some Blocking IO Operations like interacting with Database, File System, External Services then it will follow different approach
    4. Checks Threads availability from Internal Thread Pool
    5. Picks up one Thread and assign this Client Request to that thread.
    6. That Thread is responsible for taking that request, process it, perform Blocking IO operations, prepare response and send it back to the Event Loop
    7. Event Loop in turn, sends that Response to the respective Client.



Don’t Block the Event Loop (aka the Main Thread)

Like most solutions, there are advantages and disadvantages, and Node.js is not an exclusion of this. Since we know Node.js runs using the event loop, aka as the main thread, blocking the loop will indeed prevent the system from running other instructions regardless of whether they belong to a single process or multiple different processes.

If our Node.js application is the one using intensive CPU processing power to execute power, it means we cannot execute other sets of instructions until the heavy processing power instruction completes. **This is called blocking the event loop.**